SOFTWARE ENGINEERING LABORATORY (SEL)

CLEANROOM PROCESS MODEL

NOVEMBER 1991

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES SEL-91-004

SOFTWARE ENGINEERING LABORATORY (SEL)

CLEANROOM PROCESS MODEL

NOVEMBER 1991

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1976 and has three primary organizational members:

NASA/GSFC, Systems Development Branch

University of Maryland, Department of Computer Science

Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The major contributor to this document is

Scott Green (NASA/GSFC)

Additionally, the following persons contributed significantly:

Victor Basili (University of Maryland)
Sally Godfrey (NASA/GSFC)
Frank McGarry (NASA/GSFC)
Rose Pajerski (NASA/GSFC)
Sharon Waligora (Computer Sciences Corporation)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This document describes the Software Engineering Laboratory (SEL) cleanroom process model. The model is based on data and analysis from previous cleanroom efforts within the SEL and is tailored to serve as a guideline in applying the methodology to future production software efforts. It describes the phases that are part of the process model life cycle from the delivery of requirements to the start of acceptance testing. For each defined phase, a set of specific activities is discussed, and the appropriate data flow is described. This document also presents pertinent managerial issues, key similarities and differences between the SEL's cleanroom process model and the standard development approach used on SEL projects, and significant lessons learned from prior cleanroom projects. It is intended that the process model described in this document will be further tailored as additional SEL cleanroom projects are analyzed.

Table of Contents

Table of Contents (Cont'd)

Table of Contents (Cont'd)

List of Illustrations

SECTION 1—INTRODUCTION

This document describes the cleanroom process model to be used on Flight Dynamics Division (FDD) software systems, based on results and analysis of the cleanroom methodology as applied to previous FDD software development efforts. The process model described in this document is a combination of actual activities applied on earlier efforts and suggested enhancements for future projects. The information presented here should serve as a guideline in applying the methodology to projects within the FDD. However, this document is intended for use as a supplement to organized training in the methodology before any cleanroom project is initiated, and not as a replacement for appropriate training.

This document is primarily targeted for software developers and project leaders on cleanroom projects and is appropriate for Goddard Space Flight Center (GSFC) and contractor personnel within the Software Engineering Laboratory (SEL). Additionally, recommended SEL software development procedures should be followed when a specific description is not contained in this document (References 1 and 2). Updates to the process model, and therefore to this document, are expected as further analysis is performed and additional development efforts are completed.

1.1 DOCUMENT OVERVIEW

Section 1 describes the document's purpose and intended audience, and supplies background information for the remainder of the document. Section 2 describes the key similarities and differences between the SEL cleanroom model and the standard SEL development approach. Section 3 outlines some specific managerial and configuration control activities that need to be addressed during a SEL cleanroom project. Section 4 contains a summary of significant lessons learned from past cleanroom development efforts. Section 5 presents the detailed cleanroom process model and provides pictorial representations of the technical components in that process model. A glossary, list of references, and appendix outlining the steps in box structure expansion are also included.

1.2 THE SOFTWARE ENGINEERING LABORATORY

The SEL is sponsored by the National Aeronautics and Space Administration (NASA)/ GSFC to investigate the effectiveness of software engineering technologies when applied to the development of applications software (Reference 3). It was organized in 1976 with the following goals:

1. To understand the software development process in a particular environment
2. To measure the effects of various development techniques, models, and tools on this development process
3. To identify and apply improved methodologies in the GSFC environment.

The principal participants in the SEL are the Systems Development Branch of the FDD of NASA/GSFC, the Department of Computer Science of the University of Maryland, and the Systems Development Operation of Computer Sciences Corporation. Over the past 15 years, the SEL has investigated numerous techniques and methods involving dozens of projects in order to understand and improve the software development process in the FDD environment.

1.3 THE CLEANROOM METHODOLOGY

The cleanroom methodology was selected for study in the SEL for a variety of reasons (Reference 4). The SEL was interested in optimizing the software development process and improving the effectiveness of software testing, thereby reducing the rework effort that encompasses a significant portion of the FDD development effort (Reference 5). The cleanroom methodology also displayed the potential to increase software quality and reliability without negatively affecting productivity, an area of interest in any software environment. Additionally, the cleanroom methodology presented an opportunity to increase the SEL's understanding in the application of a development methodology with previously limited exposure in the software community.

The cleanroom software development methodology was conceived in the early 1980s by Dr. Harlan Mills while at IBM (References 6–10). The term "cleanroom" originates in the integrated circuit (IC) production process, where ICs are assembled in dust-free "clean rooms" to prevent the destructive effects of dust. When applying the cleanroom methodology to the development of software systems, the primary focus is on software defect prevention rather than defect removal.

The goal of cleanroom software engineering is to incrementally build an error-free product through statistical quality control. The essential elements stress the use of a structured development approach and include an emphasis on human discipline in the software process. These elements are enforced in four significant areas. First, there is a distinct separation of specific development and test activities. Second, developers rely on peer review and code reading as the primary verification techniques. Third, the purpose of testing is for quality assessment rather than for debugging or detection of defects. Finally, a top-down development approach with emphasis on incremental builds is followed to allow early assessment of product quality. Figure 1-1 illustrates the SEL cleanroom build process.

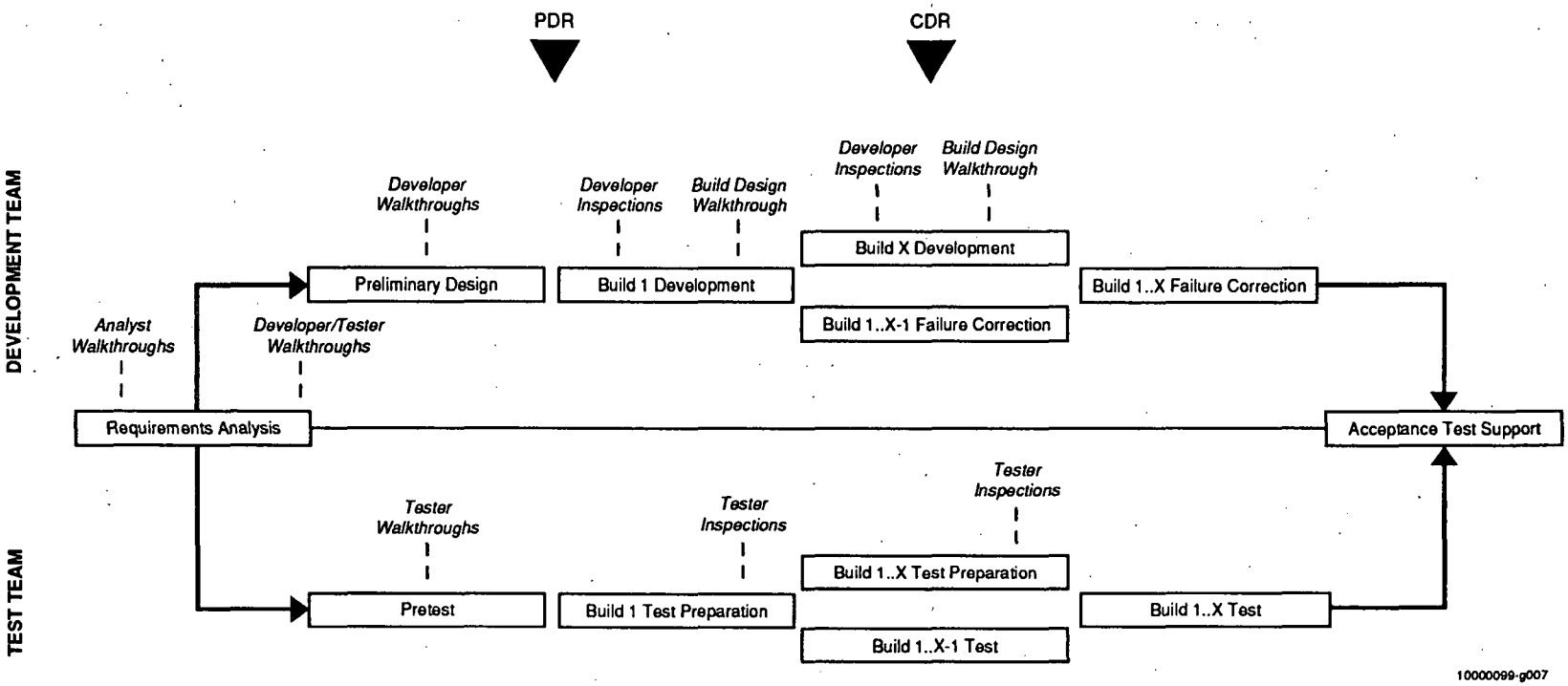

Figure 1-1. SEL Cleanroom Build Process

SECTION 2—RELATIONSHIP TO THE TRADITIONAL SEL APPROACH

This section describes some important similarities and differences between the SEL's cleanroom process model and the traditional development approach used on SEL projects.

2.1 KEY SIMILARITIES

The products and activities contained in this section are similar to items found in the SEL's *Recommended Approach to Software Development* (Reference 1) and *Manager's Handbook for Software Development (Revision 1)* (Reference 11) and can be followed in the cleanroom process model by applying the minor adjustments noted.

2.1.1 Requirements and Specifications

The requirements definition team delivers the requirements, functional, and mathematical specifications to the cleanroom project team in the same format historically used in the FDD environment. These documents are used throughout the cleanroom process model and are not listed as specific input to individual cleanroom activities. However, they do provide a baseline for the requirements analysis, design, and pretest phases.

2.1.2 Documentation

The standard set of SEL documentation is generated for the cleanroom model, including a software development plan, requirements analysis report, user's guide, system description, and software development history report. Design documents and test plans are also produced, although the specific contents and structure may deviate from traditional SEL documentation and are based on the activities and products required as part of the model.

2.1.3 Communication With Requirements Team

As with all SEL projects, well-defined communication channels between the development and requirements teams are an essential element in the success of the software life cycle. Typically, question/answer and specification modification forms are designed to document the communication and allow information to pass between teams. These processes remain in the SEL cleanroom model and, in fact, become more significant in light of the development team's effort to always produce error-free software. Because the cleanroom process relies on stable specifications as a basis, the discipline in the process is often useful in forcing specification deficiencies into the open (Reference 8).

2.1.4 Formal Reviews

The SEL's two formal reviews—Preliminary Design Review (PDR) and Critical Design Review (CDR)—remain within the cleanroom structure, although their placement in

the life cycle is not as clearly defined since the design/code/test activities are not entirely sequential, but rather iterative within builds.

The PDR should occur at approximately the same point as in the standard SEL methodology; that is, after the high-level functional breakdown of the system has been defined. In the design phase preceding the PDR, the box structure algorithm should be used in documenting the system and subsystem structures. Generally, no code or build testing activities will have occurred by PDR, although a build content schedule should be in place.

The CDR should be viewed as a status update, occurring after the iterative development activities have begun. The review should be scheduled sometime between the delivery of the first and second builds to the test team. Since the total system design will not be complete at this time, the contents of the review should focus on management and build status rather than on the detailed technical design of the system. However, open design issues should be indicated and their impact addressed.

As a complement to the traditional SEL reviews, additional but less formal Build Design Reviews (BDRs) should be scheduled during the design and implementation of each build. Whereas the CDR targets its material to the management level, a BDR should focus on the specific detailed design material, with the requirements team analysts as the intended audience. A BDR should be scheduled as the build process shifts from box structure design and review into the actual coding activities for the build. This allows the requirements team to review the design with minimal impact to the software product, thereby reducing rework for the development team.

2.2 KEY DIFFERENCES

The items contained in this section highlight the primary differences between the SEL cleanroom process model and the traditional SEL development approach.

2.2.1 Separate Teams

Unlike the traditional SEL project team, the cleanroom project team is divided into two distinct teams: a development team and a test team. The development team is responsible for designing and coding the software; the test team handles all integration and system testing activities. However, the two teams work together as a single unit in analysis of the requirements and in support of acceptance testing.

2.2.2 Computer Access

The development team completes all design and implementation of the system without access to the target computer. The software is passed to the test team after the developers complete desktop verification; the test team handles all compilation and integration activities. No unit testing occurs within the cleanroom process model.

Although coding on the target machine is prohibited, the use of personal computers (PCs) and text files is encouraged as a mechanism for recording the code and uploading it to the testers.

2.2.3 Box Structure Design

The introduction of box structures as proposed by Mills (References 6 and 10) provides a formal design method for the development team to document and verify the software design. Box structure design is a stepwise refinement and verification process that describes components in three forms: black boxes, state boxes, and clear boxes.

A black box provides an external view of the component based on data abstraction usage. A state box provides an intermediate internal view of the component based on internal state and the use of internal data abstractions. A clear box provides a more detailed internal view in which sequential or concurrent usage of additional data abstractions replaces the state box's internal data. The clear box view may also indicate the need for decomposition of additional black boxes.

2.2.4 Software Verification

Since access to the target computer and compiler is denied, the development team must rely on team reviews and code reading as the key techniques for software verification. These techniques replace unit testing as the guide for determining when the software components should be placed under configuration control.

2.2.5 System Testing Approach

The evolution of the cleanroom testing approach (Reference 9) is based on a goal of verifying reliability rather than detecting errors. System testing in the cleanroom process model is based on a statistical profile of how the system will be used operationally, rather than the traditional SEL functional testing approach based on system capabilities. By testing according to operational concepts, paths of the system used most frequently will be tested most thoroughly. Thus, the amount of time spent in testing various paths will directly correspond to the expected use of the system.

However, situations may occur where it is necessary to ensure adequate testing of system paths labeled as critical or essential, even though the operational distribution may indicate a low likelihood of expected use. The cleanroom process model does not prohibit targeted testing on these areas, but carefully removes these test scenarios from any statistical analysis.

Cleanroom testing is also driven by the fact that the development team delivers the software in increments. Although all tests are designed to execute the entire system, the successive addition of capabilities governs the functions available for execution. Therefore, the complete system is not actually tested until the delivery of the final build from the development team.

2.2.6 Acceptance Test Support

During acceptance testing, the development and test teams work together to resolve software anomalies. The development team is responsible for making all software

changes and verifying their correctness. The test team is responsible for confirming that the operational scenario in which any failure was found can be executed successfully. The test team also continues to handle configuration control of the system, downloading components upon request to the development team for correction. A representative from both teams should attend acceptance test sessions and status meetings to gather information and provide preliminary assessment of problems.

SECTION 3—MANAGEMENT AND CONFIGURATION CONTROL ACTIVITIES

In addition to the technical activities performed by the development and test teams, several management and configuration control issues need to be addressed as part of the SEL cleanroom process model.

3.1 TEAM STRUCTURE DEFINITION

A decision must be made early in the life cycle concerning the staffing of the development and test teams. Staffing levels should be sufficient, and actual staffing assignments must be carefully evaluated. Attention should be given to adequate application and skill expertise in addition to previous cleanroom experience, especially when much of the project staff is unfamiliar with the methodology.

As a general guideline, development teams should have three or four members, to facilitate the peer review process. For test teams, a minimum of two testers must be assigned to allow support and review of the testing activities. On projects consisting of several subsystems, development and test team members may be assigned to multiple subsystems.

3.2 BUILD SCHEDULE DEFINITION

The build schedule should be defined before the PDR. The build schedule should outline the number of incremental builds in the development effort, the contents of each build, and the target date for delivery of the build from the development team to the test team. Since successful execution of the schedule depends on sufficient timespans for the technical teams to complete their assigned tasks, input from the team leaders should be solicited. Attention should also be given to creating builds that result in a smooth and testable pipeline of software increments. It is important that the build increments be reasonable in scope and size for the development team as well as the test team, since both work in parallel and the amount of inactive time for each must be minimized.

3.3 CONFIGURATION CONTROL PROCEDURES

Generally, the test team is responsible for defining and executing configuration control procedures in the cleanroom process model. However, input and recommendations from the project leader and development team leader should be considered when defining the procedures to be followed.

Configuration control procedures for transferring new and modified components between the teams should be documented, and decisions on the organization and layout of all required system libraries should be made. System libraries will generally be

needed for PC source code, mainframe source code, mainframe stubs, mainframe non-executable libraries (NCALs) or object code, mainframe load modules, and mainframe job control language (JCL). Configuration control procedures for adding, logging, deleting, and moving components between libraries must also be defined. All configuration control procedures should be recorded in a formal configuration management plan.

SECTION 4—LESSONS LEARNED

This section contains 10 significant lessons learned from previous SEL projects developed using the SEL cleanroom process model. The items presented here cover a wide range of process model activities and have been incorporated into the appropriate phase descriptions in Section 5.

1. The SEL cleanroom process model should be viewed as dynamic. The methodology itself is still maturing, and there may not be a concrete answer to every problem or question that arises. Since the model is being applied to production software, decisions must frequently be made to allow the project to continue toward its required completion date.

2. The cleanroom methodology relies heavily on a team-oriented development approach. Individual preferences and styles must sometimes be sacrificed in an effort to adopt standards that ease the task of understanding and reviewing peer work.

3. Concerns such as changes to specifications, application experience, and the psychological impact of the methodology on team members appear to have little or no impact on the applicability of the cleanroom process in the FDD environment.

4. A higher percentage of the total project effort is spent in design activities on cleanroom projects than is found on projects that follow the standard SEL development approach.

5. The cleanroom methodology reemphasizes the benefits of peer code review found in previous SEL studies. In addition to finding coding errors, a concentrated code reading process uncovers nonexecution faults (e.g., prologs, program design language (PDL), comments) to keep all components consistent, accurate, and up to date.

6. The testing activities used in the cleanroom model are less well-defined than the development activities. This may lead to greater experimentation and deviation from project to project with respect to the actual testing approach followed by the test team.

7. The number of components reviewed in the team inspections will probably range from 3 to 10. The variation depends on the frequency of reviews, the schedules of team members, and the size or complexity of the components being reviewed.

8. Having multiple code readers results in measurably more effective reviews than does having one reader. Previous SEL cleanroom efforts have shown

that, often, each fault is discovered by only one reader, even when multiple readers are used. Only a small percentage of the total faults are typically uncovered by all readers for any given component.

9. When examining the code phase effort, approximately the same percentage of effort is spent code reading as code writing.

10. Although a separation of team functions is essential in the cleanroom model, informal meetings and discussions between the development and test team leaders are often beneficial in isolating the true source of software failures.

SECTION 5—THE SEL CLEANROOM PROCESS MODEL PHASES

The SEL cleanroom process model is composed of five distinct phases: requirements analysis, design, code, pretest, and test. This section provides a detailed breakdown of the activities that constitute each of these phases. Each phase is further associated with the particular team responsible for performing the activities within that phase. Figure 5-1 shows the five phases and the high-level data flow among them.

The process model begins with the delivery of the requirements specifications, functional specifications, and mathematical specifications from the requirements organization. The requirements analysis phase is performed jointly by the development and test teams and serves to resolve ambiguities, incorporate corrections into the specifications, and facilitate a common discovery process in understanding the problem.

After the requirements analysis phase, the process model specifies independent activities for each team to perform. The development team begins the incremental design phase and code phase build activities, using box structure refinement to document the design process and peer code review to verify software correctness before delivery to the test team. Concurrently, the test team initiates the pretest phase by defining the statistical test approach to be followed and preparing for the development team's software build deliveries.

The structured team reviews held while designing and coding the software components are an essential part of the cleanroom process model. These peer review activities replace the traditional unit testing as the key verification technique for individual software components. **It is this strong reliance on human discipline in the software verification process that separates cleanroom from most traditional software development methodologies.**

Upon delivery of builds from the development team, the test team begins the test phase activities, generating and executing the statistical tests and validating the results. Since the development and test efforts both affect the build process, clear and concise procedures for transferring software and handling failures are required. Although the development team makes all software corrections, discussion between testers and developers is often useful in isolating the actual failure source.

The remainder of this section contains an explanation of each cleanroom process model activity, grouped according to the five phases cited. Each activity is organized into the following segments:

- *Completed by:* the team responsible for performing the activity
- *Description:* a description of the activity
- *Input Activities:* input required from other cleanroom activities
- *Output Activities:* output required for other cleanroom activities

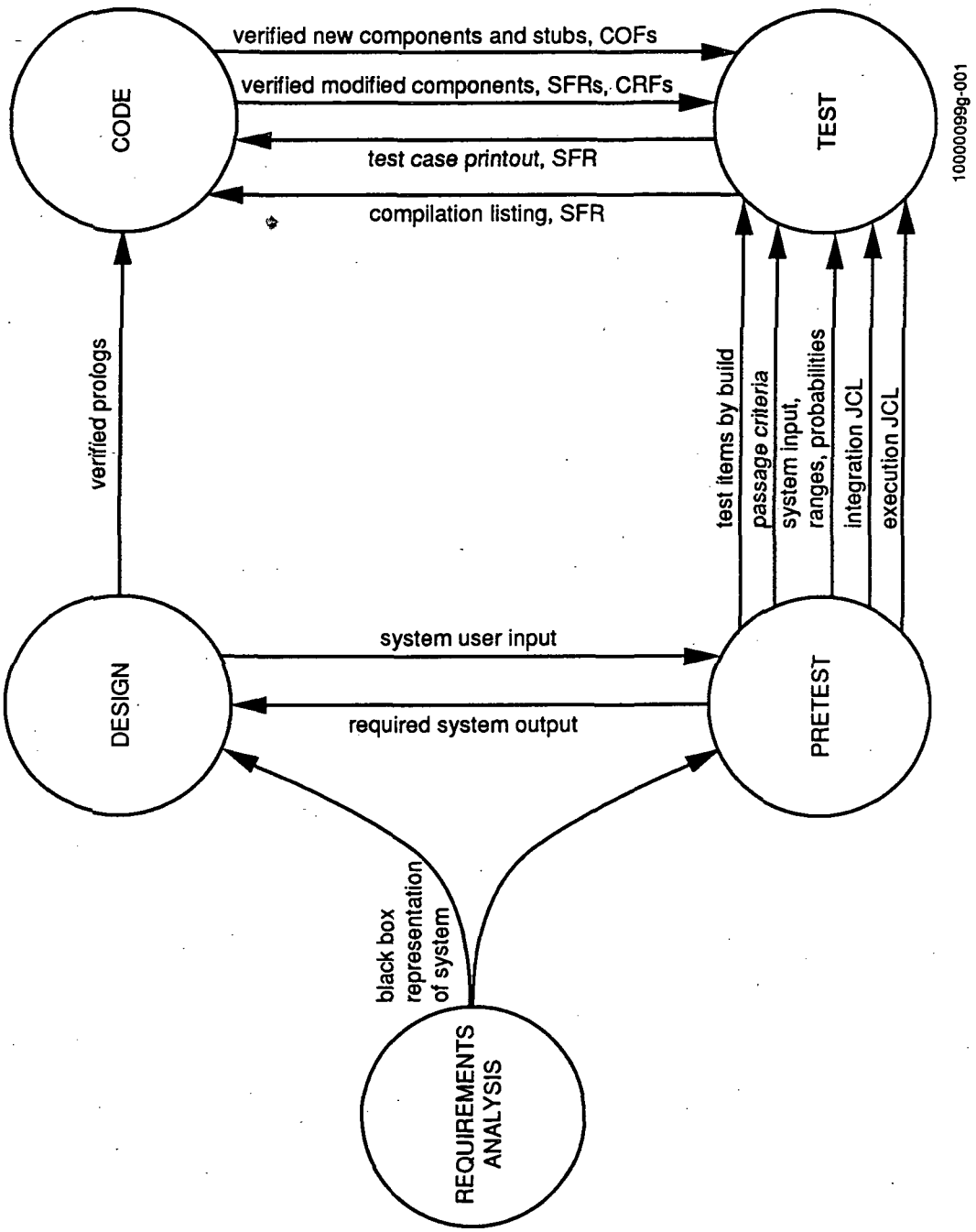

Figure 5-1. Cleanroom Process Model Phase Data Flow

- *Exit Criteria:* general questions that may assist in determining completion of the activity
- *Notes:* information or guidelines that may assist in understanding or completing the activity

5.1 REQUIREMENTS ANALYSIS PHASE

The requirements analysis phase consists of one cleanroom activity: box abstraction. The activity is performed jointly by the development and test teams to foster a common understanding of the requirements and specifications. Figure 5-2 shows the data flow resulting from requirements analysis activity.

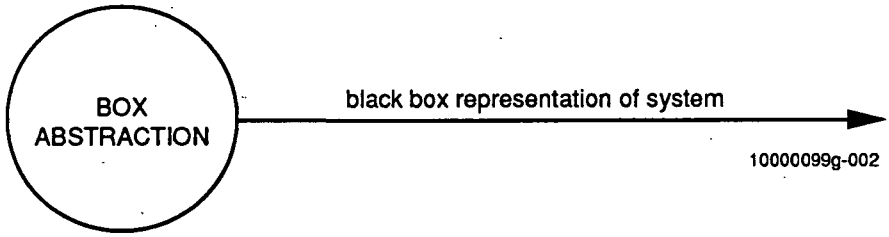

Figure 5-2. Requirements Analysis Activity

Requirements analysis begins upon delivery of the requirements specifications, functional specifications, and mathematical specifications from the requirements organization. The goal of analyzing the contents of the documents for completeness, clarity, consistency, and feasibility is similar to the traditional SEL recommended approach (Reference 1). The primary difference lies in the process used to record the analysis, namely box abstraction.

The black box representations produced in this phase are used as a baseline for the development and test teams to complete the remainder of the SEL cleanroom process model activities.

5.1.1 Activity: Box Abstraction

Completed by: Development Team and Test Team

Description: The development and test teams attend specification walkthroughs given by the requirements team and analyze the requirements specifications, functional specifications, and mathematical specifications using the following process:

- Create clear box representations of the system as follows:
  - Convert functional specifications data flows to clear box representations of the system

- Submit formal questions to account for any inconsistencies, ambiguities, or errors in specifications
- Integrate answers to formal questions and information discussed at walkthroughs into clear box representations
- Review and improve clear box representations at team meetings

- Create state box representations of the system as follows:
  - Convert clear box representations to state box representations of the system
  - Submit formal questions to account for any inconsistencies, ambiguities, or errors in specifications
  - Integrate answers to formal questions and information discussed at walkthroughs into state box representations
  - Review and improve state box representations at team meetings
- Create black box representations of the system as follows:
  - Convert state box representations to black box representations of the system
  - Submit formal questions to account for any inconsistencies, ambiguities, or errors in specifications
  - Integrate answers to formal questions and information discussed at walkthroughs into black box representations
  - Review and improve black box representations at team meetings

The teams also generate a requirements analysis report that highlights to-be-determined (TBD) items and concerns revealed through analysis of the requirements and specifications documents.

Input Activities: None

Output Activities:

- Black box refinement
  - Black box representations of the system
- User input identification
  - Black box representations of the system

- Test item description
  - Black box representations of the system
- System input identification
  - Black box representations of the system

Exit Criteria:

- Do the development and test teams have a sufficient understanding of the system to begin their respective tasks?
- Are the specifications complete enough to define the necessary box representations?
- Have all inconsistencies, clarifications, and errors been submitted through the formal question-and-answer process?
- Have development and test team members been identified by the conclusion of this activity?

Notes:

- Developers and testers work together as a single project team during this activity.
- The project team should not make any assumptions when analyzing the specifications; all input, processes, and output must be defined completely.
- The data dictionary provided in the specifications should be analyzed thoroughly for completeness.
- A formal question must be submitted for all discrepancies uncovered; reliance on verbal answers should be avoided.
- The project leader must ensure that questions, answers, and specification modifications are distributed to all team members.
- Management should expect a higher staffing level than is normally associated with comparable activities on projects following the traditional SEL development methodology.

5.2 DESIGN PHASE

The design phase consists of six activities: user input identification, black box refinement, state box refinement, clear box refinement, box structure review, and component design review. The development team performs these activities. Figure 5-3 shows the data flow for design activities.

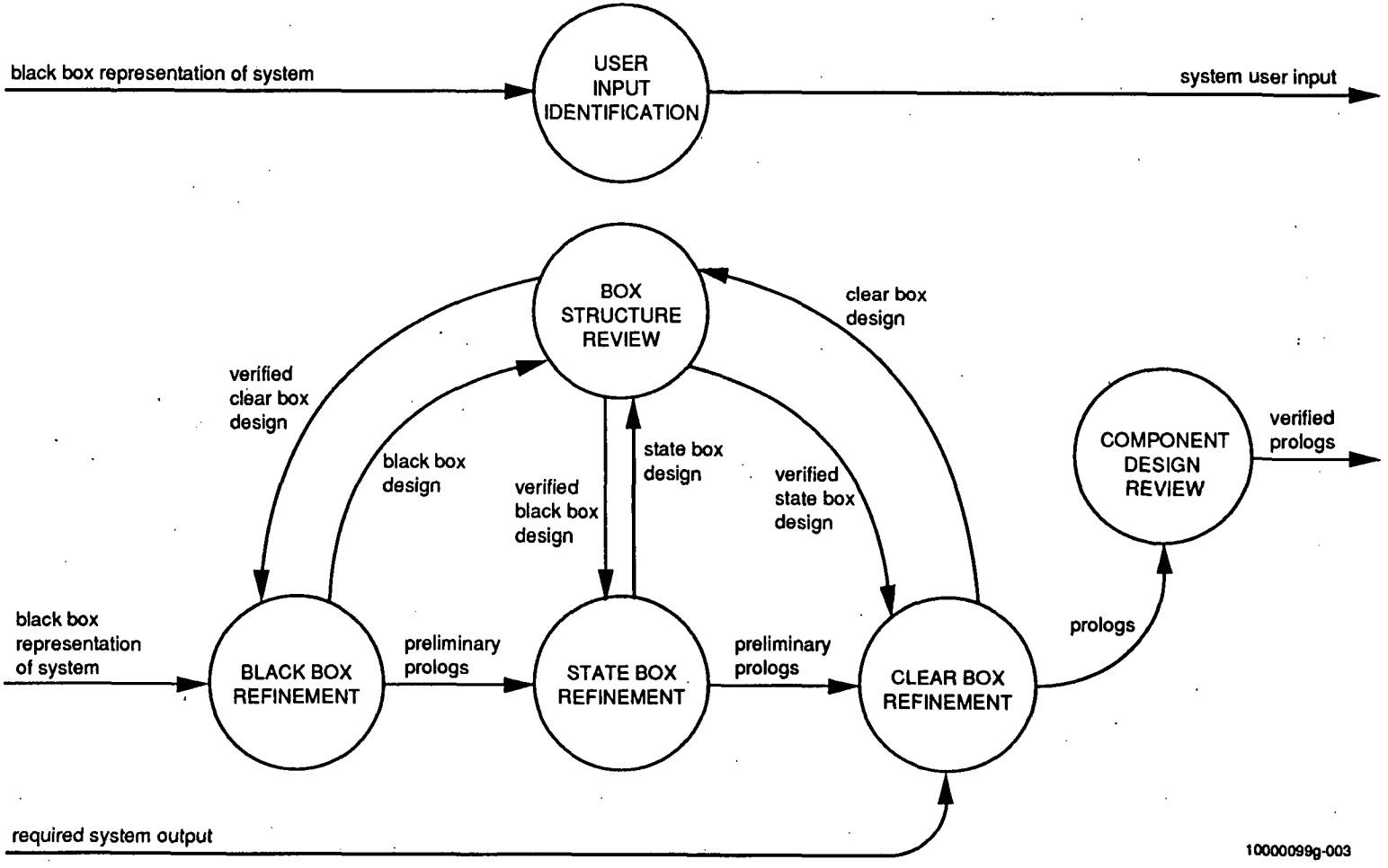

Figure 5-3. Design Activities

Input to the design phase includes the black box representations produced in the requirements analysis phase and a list of all required system output from the test team. End products include a list of all NAMELISTs and user input for the test team and the verified prologs that are used to begin the implementation process.

The black/state/clear box structure design activities used in this phase are iterative; they continue until all clear boxes are designed so that they contain no additional black boxes and all PDL has been verified.

5.2.1 Activity: User Input Identification

Completed by: Development Team

Description: The development team generates the preliminary system NAMELISTs and user input. The input is distributed to the test team to assist in its identification of all potential system input.

Input Activities:

- Box Abstraction
  - Black box representations of the system

Output Activities:

- System input identification
  - System user inputs

Exit Criteria:

- Has all user input been identified and distributed to the test team before the total input description has been completed?
- Has the development team reviewed the user input?

Notes:

- Updates to the NAMELISTs and user input should be distributed to the test team as further design is completed.
- Information gathered by the test team from discussions with the analysts should be shared with the development team.

5.2.2 Activity: Black Box Refinement

Completed by: Development Team

Description: The development team defines black boxes that will represent the various functions of the system. Specifically, the development team completes steps 1 and 2 of the box structure algorithm (see Appendix).

Functions that may not have been directly described by the specifications (e.g., high-level system drivers) need to be defined. Some black boxes may be usable from those that were generated during box abstraction. Simple functions that do not need to retain data and do not break down into lower functions may be fully designed at this point.

Additional black boxes may be identified as future clear boxes are defined, expanded, and verified. These clear boxes then become input to the black box refinement process.

Completion of black box design for a component will provide information for the following sections of the prolog:

- Module name
- Purpose
- Arguments (preliminary)
- Requirements reference
- Method/PDL (preliminary)

Black boxes need to go through a team box structure review. Baseline diagrams should be generated and updated as new components are identified.

Input Activities:

- Box abstraction
  - Black box representation of system
- Box structure review
  - Verified clear box design

Output Activities:

- Box structure review
  - Black box design
- State box refinement
  - Preliminary prologs for new components

Exit Criteria:

- Are all stimuli clearly defined for each box?
- Are all responses clearly defined in terms of stimuli for each box?
- Has the author verified the correctness of the box design?
- Has a review been scheduled?

Notes:

- Team conventions for notation should be followed when describing the black boxes.
- Most of the black boxes designed here will be different from the black box representations for the box abstraction effort.
- The black/state/clear box activities are done iteratively until a box is derived that can be mapped directly to a single component, with no remaining boxes inside it.

5.2.3 Activity: State Box Refinement

Completed by: Development Team

Description: The development team refines the black boxes into state boxes. A state box design is added to black boxes that require retained data by following steps 3, 4, 5, and 6 of the box structure algorithm (see Appendix).

State boxes are a refinement of the design black boxes, and those derived during this activity will probably be different from boxes used in box abstraction. COMMON blocks and other schema for holding state data should be defined.

Completion of state box design for a component will provide information for the following sections of the prolog:

- Arguments (updated)
- COMMON blocks referenced
- Method/PDL (updated)

State boxes need to go through a team box structure review. Baseline diagrams should also be updated as necessary.

Input Activities:

- Box structure review
  - Verified black box design
- Black box refinement
  - Preliminary prologs for new components

Output Activities:

- Box structure review
  - State box design

- Clear box refinement
  - Preliminary prologs for new components

Exit Criteria:

- Have all retained data been identified?
- Has the author verified the correctness of the box design?
- Has a review been scheduled?

Notes:

- Team conventions for notation should be followed when describing the state boxes.
- Most of the state boxes designed here will be different from the state box representations for the box abstraction effort.
- If there is no need for retained data, black and state boxes are identical.
- The black/state/clear box activities are done iteratively until a box is derived that can be mapped directly to a single component, with no remaining boxes inside it.

5.2.4 Activity: Clear Box Refinement

Completed by: Development Team

Description: The development team refines the state boxes into clear boxes by following steps 7, 8, 9, and 10 of the box structure algorithm (see Appendix).

Clear box designs should provide definitions of all local variables and complete method descriptions for the functions defined in the box structures. The clear box design should also provide all information to complete the component prolog, including:

- Arguments (updated)
- Method/PDL (updated)
- External references

The completed prolog should provide references to all system output as required by the test team for validation of test items.

After clear boxes have been expanded, additional black boxes may need to be defined. Step 11 of the box structure algorithm should be followed for these expanded black boxes (see Appendix).

All clear boxes can be mapped directly to component prologs and PDL, which should be verified through a component design review. The baseline diagrams should be modified to reflect the addition of any new black boxes.

Input Activities:

- Box structure review
  - Verified state box design
- State box refinement
  - Preliminary prologs for new components
- Output data identification
  - Required system output

Output Activities:

- Box structure review
  - Clear box design
- Component design review
  - Prologs for new components

Exit Criteria:

- Is all information needed for completion of the prolog available?
- Has the author verified the correctness of the design?
- Have any additional black boxes derived from this clear box been identified?
- Has a review been scheduled?

Notes:

- Team conventions for notation should be followed when describing the clear boxes.
- Most of the clear boxes designed here will be different from the clear box representations for the box abstraction effort.
- The black/state/clear box activities are done iteratively until a box is derived that can be mapped directly to a single component, with no remaining boxes inside it.

5.2.5 Activity: Box Structure Review

Completed by: Development Team

Description: The development team reviews the box structures to understand the system design, identify potential problems, and resolve ambiguities. The following process is used:

1. The box structure designs are distributed to the reviewers a few days before the scheduled review.
2. Reviewers prepare for the review by independently studying the material to be presented.
3. At the review, the box structures are examined, and questions and comments are discussed. All problems should be noted on a master copy of the design materials.
4. A decision is made at the end of the review on the necessity of a rereview of the material.
5. The design is corrected by the author and reviewed again, if necessary.

Input Activities:

- Black box refinement
  - Black box design
- State box refinement
  - State box design
- Clear box refinement
  - Clear box design

Output Activities:

- State box refinement
  - Verified black box design
- Clear box refinement
  - Verified state box design
- Black box refinement
  - Verified clear box design

Exit Criteria:

- Have reviewers decided whether another review is necessary?
- Have all corrections or changes been noted on a master copy of the design material being reviewed?
- Have all changes and corrections been incorporated into the design?

Notes:

- It may be convenient in some instances to review black and state boxes together.
- Although box structure reviews are intended to be less formal than component design reviews or component code reviews, their importance should not be diminished.

5.2.6 Activity: Component Design Review

Completed by: Development Team

Description: The development team reviews component prologs and PDL to confirm functionality and identify errors, using the following process:

1. The component prologs and PDL are distributed to the reviewers a few days before the scheduled review.
2. The reviewers prepare for the review by independently studying the material to be presented.
3. At the review, the component designs are examined, and questions and comments are discussed. All errors and action items should be noted on a Component Status Form.
4. A decision is made at the end of the review on the necessity of a rereview of the material.
5. The design is corrected by the author and reviewed again, if necessary.

Input Activities:

- Clear box refinement
  - Prologs for new components

Output Activities:

- Component implementation
  - Verified prologs for new components

Exit Criteria:

- Have reviewers decided whether another review is necessary?
- Have all changes, corrections, and actions been noted on the data collections forms?
- Have all changes and corrections been incorporated into the prologs and PDL?
- Does the development team have sufficient information to begin coding?

Notes:

- Emphasis during these reviews is on fault isolation rather than fault correction.
- In addition to the author, there should be at least two other reviewers for each set of material.
- It may be useful to keep a marked-up master copy of the component prolog and PDL during the review.
- Each component prolog and PDL will probably require two or three reviews.
- All faults should be noted on Component Status Forms, including cosmetic errors.
- Any person on the development team should be able to begin coding a component once it has passed this review stage; each reviewer should review as though he or she will be the person required to code the component.
- Every reviewer must be able to completely understand the contents of the prolog; a single reviewer's objection is sufficient to force some rework to the component.

5.3 CODE PHASE

The code phase consists of four activities: component implementation, fault isolation, component modification, and component code review. The development team performs these activities. Figure 5-4 shows the data flow between the code activities.

Inputs to the code phase include the verified prologs from the Design phase, along with Software Failure Reports (SFRs) and associated testing documentation from the test team. End products include new and modified components, verified for correctness by the development team.

The development team uses the verified prologs to generate the corresponding software components. The development team is also responsible for isolating faults in

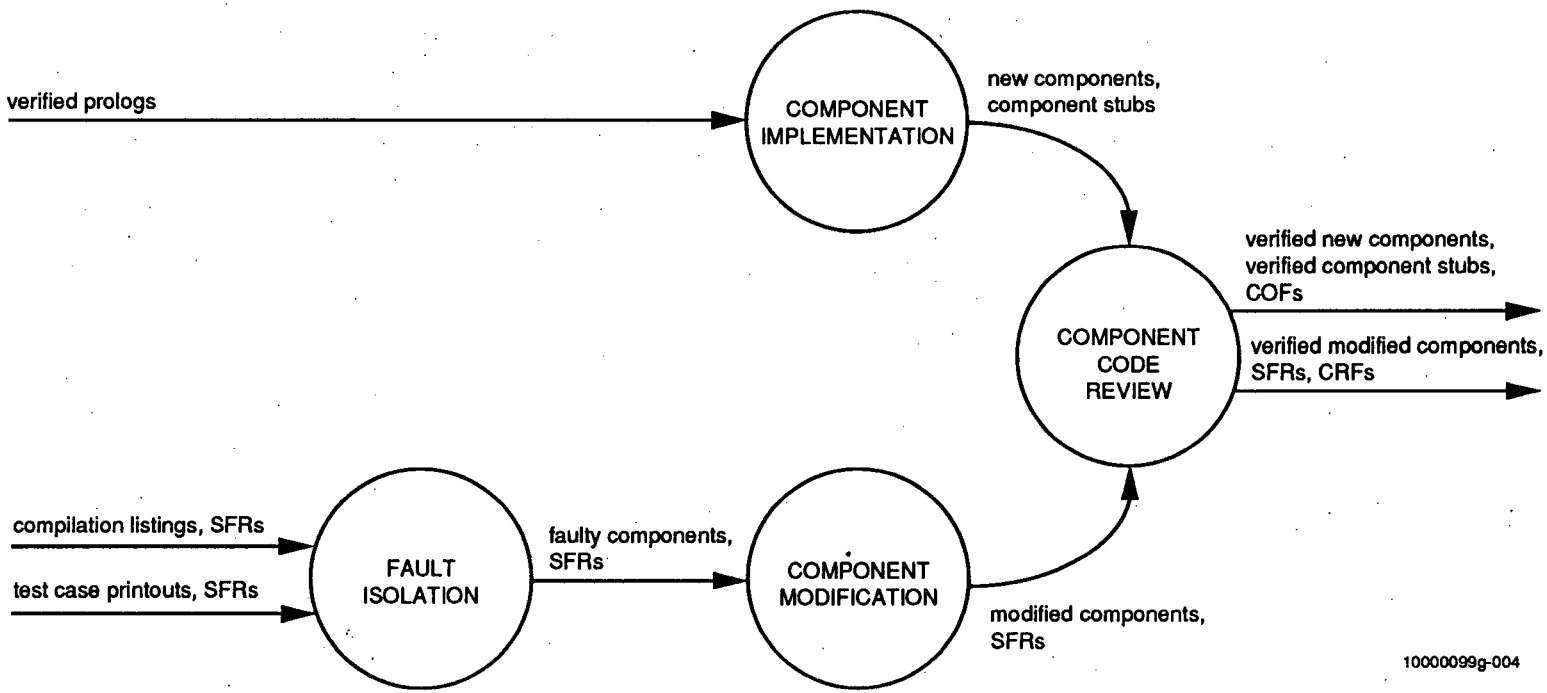

Figure 5-4. Code Activities

previously delivered builds and modifying the appropriate components. All components must pass through a code review process before being delivered to the test team.

5.3.1 Activity: Component Implementation

Completed by: Development Team

Description: The development team writes code for each component, based on the verified prologs and PDL. Components should be coded according to the build schedule and the planned delivery of builds to the test team. All required component stubs that are needed for testing of the build must also be generated.

Input Activities:

- Component design review
  - Verified prologs for new components

Output Activities:

- Component code review
  - New components
  - Component stubs

Exit Criteria:

- Have the components been coded according to the team's coding standards?
- Has the developer/author examined the code for correctness before passing it to reviewers?
- Have all required component stubs been written?

Notes:

- Code must be written so that it is easy to follow, understand, and maintain by someone other than the author.
- Individual coding styles should be eliminated in favor of common team and SEL coding guidelines.

5.3.2 Activity: Component Code Review

Completed by: Development Team

Description: The development team reviews the newly coded or modified components to confirm implementation of the design and identify potential faults, using the following process:

1. Component listings are distributed to the reviewers a few days before the scheduled review.
2. Reviewers prepare for the review by independently code-reading the material to be presented.

3. At the review, the components are examined, and questions and comments are discussed. All errors and action items should be noted on a Component Status Form.

4. A decision is made at the end of the review on the necessity of a rereview of the material.

5. The code is corrected by the author and reviewed again, if necessary.

After components have been verified, the appropriate SEL forms are completed: Component Origination Forms (COFs) for new components and Change Report Forms (CRFs) and SFRs for modified components.

Input Activities:

- Component implementation
  - New components
  - Component stubs
- Component modification
  - Modified components
  - Software Failure Reports

Output Activities:

- System integration
  - Verified new components
  - Verified component stubs
  - Verified modified components
  - COFs
  - CRFs
  - SFRs

Exit Criteria:

- Have reviewers decided whether another review is necessary?
- Have all changes, corrections, and action items been noted on the data collection forms?
- Have all changes and corrections been incorporated into the software components?

- Has a COF been completed for each new component?
- Have all necessary CRFs and SFRs been completed for modified components?

Notes:

- A master listing of the component should be kept during the review by a designated recorder, to consolidate all necessary changes.
- Each component should be read by at least two developers other than the original author.
- Code should be clear, complete, and easily understood by all reviewers; this is the last chance for the development team to review the components before they are delivered to the test team.
- Prologs and PDL should be checked for updates during the review of modified components.
- Changes and corrections should be viewed as constructive and team oriented, not antagonistic.
- Each reviewer should establish a systematic way of reading (e.g., first checking variables, then calling sequences) to avoid oversights. Since developers will probably have unique styles and therefore differing strengths in their reading approach, the chance of an error's getting through multiple reviewers is reduced.
- Code reading by stepwise abstraction is strongly recommended.
- If changes are other than cosmetic, the corrected components should be redistributed and reviewed again.

5.3.3 Activity: Fault Isolation

Completed by: Development Team

Description: The development team receives an SFR and appropriate compilation/link listing or test case printout from the test team. The development team uses the information to isolate the cause of the software failure. When all software components requiring correction have been identified, the development team requests the test team to download the components for modification.

Input Activities:

- System integration
  - Compilation/link listing
  - SFR

- Test case validation
  - Test case printout
  - SFR

Output Activities:

- Component modification
  - Faulty components
  - SFRs

Exit Criteria:

- Has the source of the failure been isolated?
- Have all components requiring modification been requested and received from the test team?
- Have proper configuration control measures and procedures been followed to note which components are being modified by the development team?

Notes:

- The failure source should be carefully isolated before any components are requested for modification.
- Up-to-date bluebook listings should be kept to aid in the fault isolation process.
- The development team should initially attempt to isolate the failure source by examining the design and test case printout. However, it may become necessary to request additional diagnostic data to assist in tracking the problem.
- Informal discussions with the test team may be useful in gathering information about the failure.

5.3.4 Activity: Component Modification

Completed by: Development Team

Description: The development team modifies the requested components after receiving them from the test team.

Input Activities:

- Fault Isolation
  - Faulty components
  - SFRs

Output Activities:

- Component Code Review
  - Modified components
  - SFRs

Exit Criteria:

- Have the components been modified according to the team's coding standards?
- Has the developer/modifier examined the code for correctness before passing it to others for review?
- Have all necessary prologs and PDL been updated?

Notes:

- Corresponding prolog and PDL updates must be made whenever a component is modified.
- Team coding standards should be followed during all modifications.

5.4 PRETEST PHASE

The pretest phase consists of five activities: test item description, input description, output data identification, test item mapping, and JCL generation. The test team performs these activities. Figure 5-5 shows the data flow for pretest activities.

Input includes the black box representations from the requirements analysis phase and the list of system input from the development team's design efforts. End products include a list of required system output for the development team and a description of all test items, passage criteria, system input values and probabilities, and JCL to be used in the testing process.

The goal of the pretest phase is to prepare the test team for the beginning of testing activities upon delivery of the first build from the development team. By the conclusion of these activities, the test team should have a clear understanding of most system input and system operational scenarios.

5.4.1 Activity: Test Item Description

Completed by: Test Team

Description: The test team identifies all test items and the passage criteria to be used for each. The specifications are examined to identify significant and testable elements. These may include output from specific equations, values of intermediate parameters

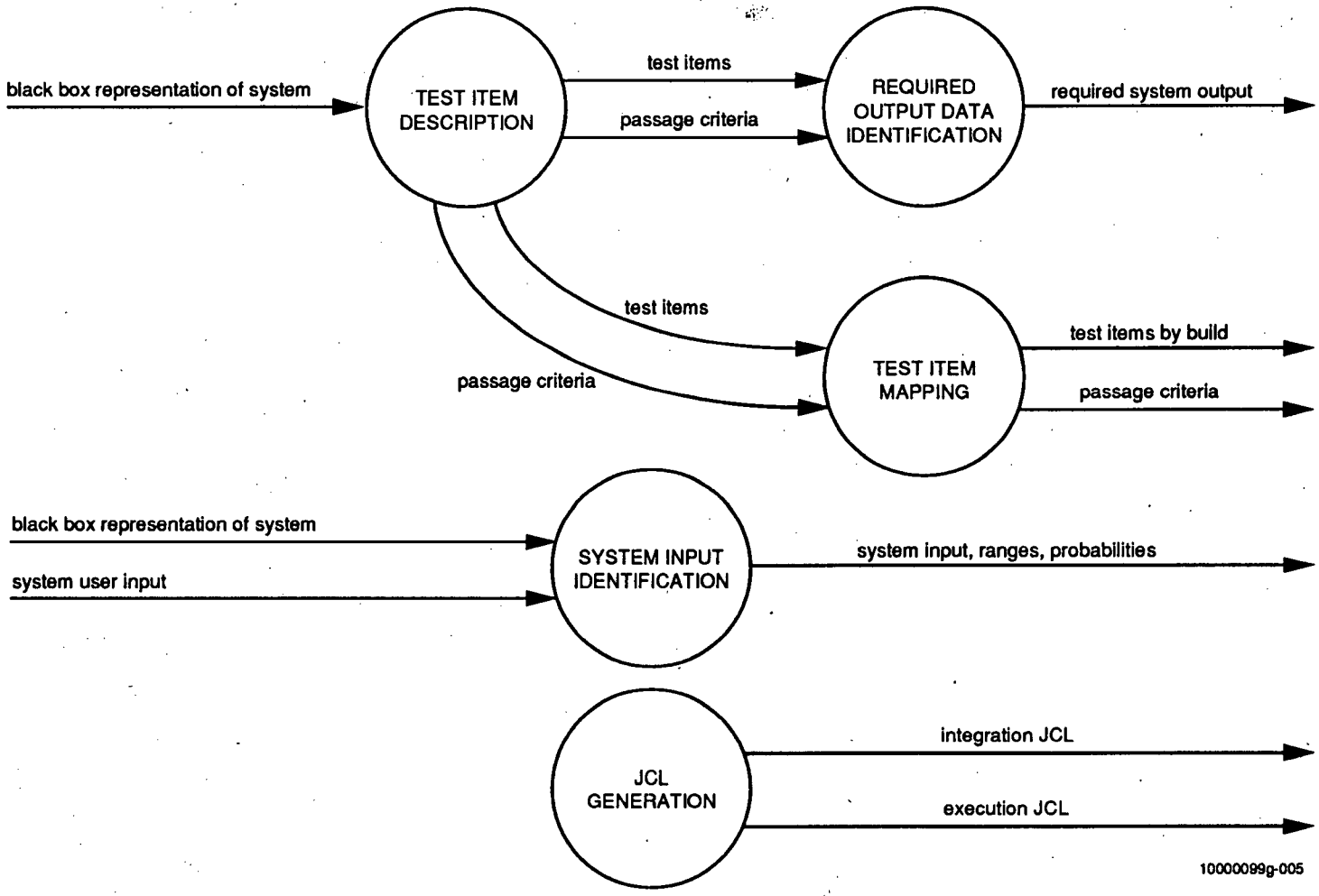

Figure 5-5. Pretest Activities

in equations, entire subfunctions (e.g., the reading of NAMELISTs), and options or displays that are part of the user interface. The test team must also define passage criteria to serve as a guideline in validation descriptions of each specific test item. Multiple testers should review the functional specifications, the list of test items, and the passage criteria to ensure correctness, feasibility, and complete system coverage.

Input Activities:

- Box abstraction
  - Black box representations of the system

Output Activities:

- Test item mapping
  - Test items
  - Passage criteria
- Required output data identification
  - Test items
  - Passage criteria

Exit Criteria:

- Has the list of test items been reviewed to ensure complete system coverage?
- Have the passage criteria for each test item been reviewed by another tester for completeness, correctness, and feasibility?

Notes:

- The test team must ensure that all pertinent test items are specified; this list of test items will be the foundation for the test team's activities throughout the testing process.
- As modifications are made to the specifications, additional test items may need to be identified.

5.4.2 Activity: System Input Identification

Completed by: Test Team

Description: The test team determines the operational profile of the system, identifies all potential input to the system, and determines the possible ranges for each input value. Probabilities of selection are assigned to each value or set of values in the range,

based on the system operational scenarios. These distributions are used in the statistical test case generation.

Input Activities:

- Box abstraction
  - Black box representations of systems
- User input identification
  - System user input

Output Activities:

- Test case generation
  - System input
  - Ranges for system input
  - Probabilities for system input values

Exit Criteria:

- Has all potential system input been identified?
- Have distribution ranges and associated probabilities been assigned for each system input?

Notes:

- All resources should be examined in understanding the operational profile of the system: interviews with operators and analysts, demonstrations of similar systems, personal experiences, specification documents, and operational concepts documents.
- The information gathered by the test team may also be beneficial to the development team in its understanding of how the system should function.
- Testers should submit formal questions to document concerns whenever it is unclear to them how the system will actually be used operationally.

5.4.3 Activity: Test Item Mapping

Completed by: Test Team

Description: The test team maps each test item to the earliest build where that test item can be tested and validated. When major or critical test items cannot be fully validated until late builds, intermediate elements may become new test items with passage criteria of their own.

Input Activities:

- Test item description
  - Test items
  - Passage criteria

Output Activities:

- Test case generation
  - Test items organized by builds
  - Passage criteria
- Test case validation
  - Test items organized by builds
  - Passage criteria

Exit Criteria:

- Do testers have a complete list of test items organized by builds?
- Is each test item mapped to a single build?

Notes:

- The complete list of test items organized by builds is one of the primary elements used by the test team during the actual testing process.
- Passage criteria should be defined as new test items are identified.

5.4.4 Activity: Required Output Data Identification

Completed by: Test Team

Description: The test team identifies all data that must be available as output from the system to facilitate the testing process. The list of required output data is passed to the development team and should include all output needed for verification and validation of the specified test items.

Input Activities:

- Test item description
  - Test items
  - Passage criteria

Output Activities:

- Clear box refinement
  - Required system output

Exit Criteria:

- Has the test team thoroughly analyzed what data need to be available to verify all test items?
- Has the test team submitted the list of necessary output in time for the development team to include it in the appropriate box refinements?

Notes:

- The output information should be passed on to the development team as early as possible to minimize rework on its part; updates to the list should be passed over as soon as they are known.
- The development team may make the required output available in many different formats (e.g., displays, reports, and debug).
- If the same data item is required at different points in the processing stream, it should be listed as a separate item each time it is needed.
- The list of output data generated in this activity is based solely on the needs of the test team and may be different from the system output listed by the requirements team in the functional specifications.

5.4.5 Activity: JCL Generation

Completed by: Test Team

Description: The test team generates all JCL required to compile, link, and execute the test cases.

Input Activities: None

Output Activities:

- System integration
  - Integration JCL
- Test case execution
  - Execution JCL

Exit Criteria:

- Has all required JCL been generated?
- Has all required JCL been reviewed by another member of the test team?

Notes: None

5.5 TEST PHASE

The test phase consists of four activities: system integration, test case generation, test case execution, and test case validation. The test team performs these activities. Figure 5-6 shows the data flow between the test activities.

Input from the pretest phase includes the defined set of test items, passage criteria, and system input descriptions. These are used in the generation and validation of test cases. Verified software components and associated SEL forms are input from the development team's coding effort for integration of the system. The required integration and execution JCL are also input from the pretest activities. The primary end products from the phase are the SFRs generated when failures occur in the integration or execution process. The test team transfers these reports, along with any other pertinent information, to the development team for fault isolation and correction.

5.5.1 Activity: System Integration

Completed by: Test Team

Description: For new build deliveries, the development team delivers components, COMMON blocks, BLOCK DATAs, NAMELISTs, and necessary component stubs to the test team, along with the corresponding COF. The development team places the new source code into the PC system library. The test team then uploads the new code to the mainframe system library and uploads the required stubs to the mainframe stubs library. The test team logs components using COF information and submits the forms to the SEL database personnel.

After modifying faulty components, the development team delivers components, COMMON blocks, BLOCK DATAs, and NAMELISTs downloaded for correction back to the test team, along with the corresponding CRFs and SFRs. The development team places the modified source code into the PC system library. The test team then uploads the modified code to the mainframe system library. The test team records the appropriate information from the CRFs and SFRs and submits the forms to the SEL database personnel.

The test team creates and submits compilation JCL for the appropriate components. An SFR is generated for each component that does not compile cleanly, and those components are transferred back to the development team. When all components and component stubs required for a given build have been successfully compiled, the test team creates and submits JCL to create a load module. An SFR is generated for any failure that can be traced to a software component.

Input Activities:

- Component code review
  - Verified new components
  - Verified component stubs

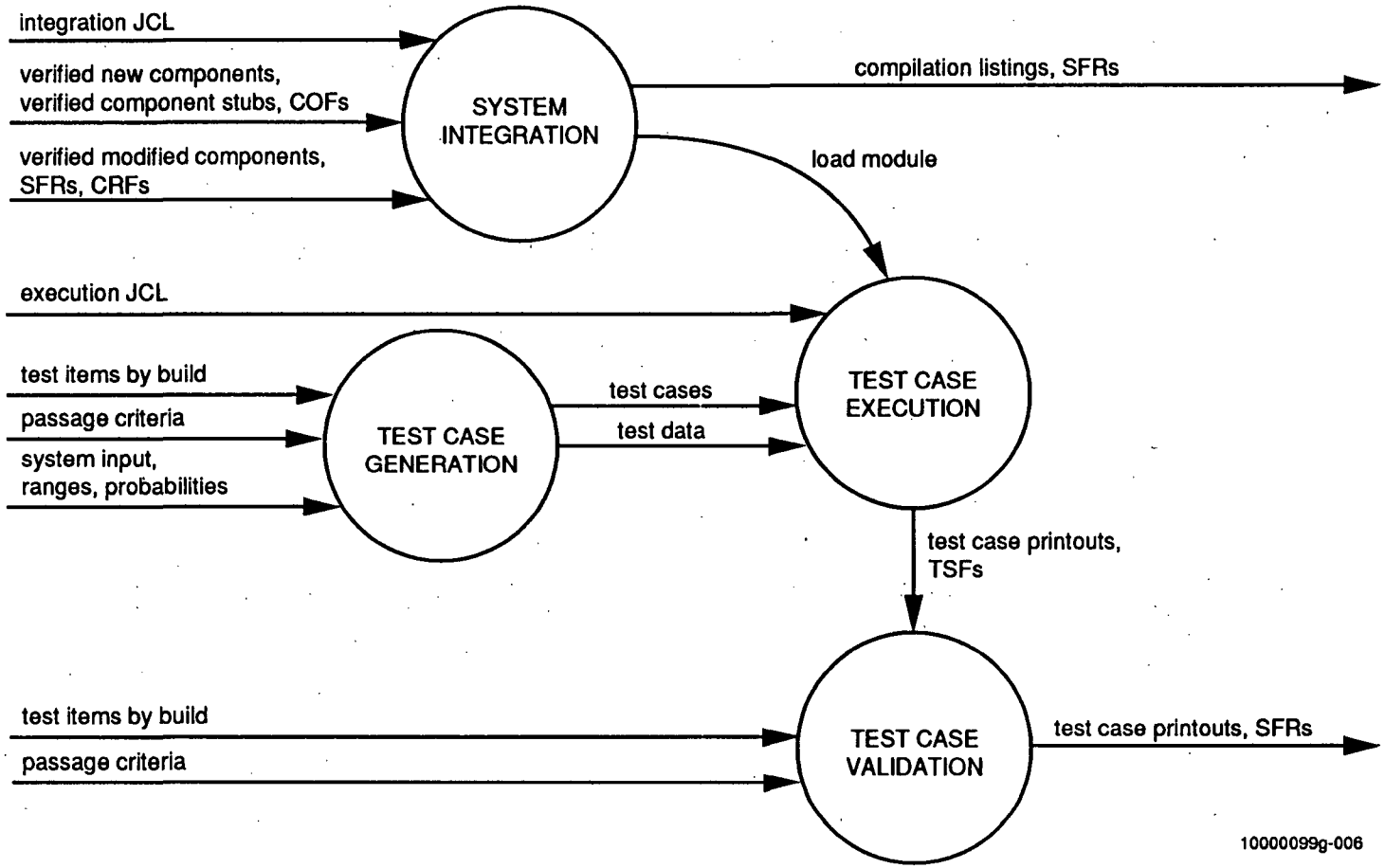

Figure 5-6. Test Activities

  - Verified modified components
  - COFs
  - CRFs
  - SFRs
- JCL generation
  - Integration JCL

Output Activities:

- Fault isolation
  - Compilation listing
  - SFR
- Test case execution
  - Load module

Exit Criteria:

- For new source code, does the delivery represent a complete build?
- Have all COFs been included with the delivery?
- Have all required component stubs been included with the delivery?
- For modified source code, have all CRFs and SFRs been included with the delivery?
- Have all components downloaded to the development team for modification been accounted for?
- Have all components compiled cleanly?
- Have SFRs been generated for all components with compilation errors?
- Has a load module been created without software component errors?

Notes:

- The development team corrects all compilation errors and warnings.
- If a component replaces a previously delivered component stub, the test team assures that the corresponding stub has been deleted from the stubs library.

5.5.2 Activity: Test Case Generation

Completed by: Test Team

Description: The test team determines how many test cases are needed for each build to satisfy the validation of test items associated with the build. For statistically generated test input values, a test case generator, random number generator, or data simulator can be used to generate actual test case values. Internal team peer inspections should be used after a set of test cases has been generated to assure completeness.

After examining the statistically generated test cases, a decision must be made regarding specific test items that are considered critical but are not sufficiently exercised in the existing set of test cases. If additional test cases are desired, they may be generated without reliance on the statistical probabilities for input ranges. However, these nonstatistically generated test cases should be noted as such, and the results should not be included in mean-time-to-failure (MTTF) evaluations.

Input Activities:

- System input identification
  - System input
  - Ranges for system input
  - Probabilities for system input values
- Test item mapping
  - Test items organized by builds
  - Passage criteria

Output Activities:

- Test case execution
  - Test cases
  - Test data

Exit Criteria:

- Are test cases for the build complete and ready to be executed?
- Have all proposed test cases been reviewed by another test team member?

Notes:

- Only statistically generated test cases should be used in MTTF evaluations.
- The test team should use nonstatistically generated tests judiciously; they should be reserved for critical test items that the test team feels must be

evaluated without regard to their probable occurrence in the program's operational scenarios.

5.5.3 Activity: Test Case Execution

Completed by: Test Team

Description: The test team executes the test case and initiates a corresponding Test Status Form (TSF). All pertinent information is recorded on the TSF, including any unplanned deviation from the defined test.

Input Activities:

- Test case generation
  - Test case
  - Test data
- JCL generation
  - Execution JCL
- System integration
  - Load module

Output Activities:

- Test case validation
  - Test case printout
  - TSF

Exit Criteria:

- Has the test case been executed successfully, without test team error?
- Has all pertinent information been included on the TSF?

Notes:

- The test team should keep all test-related information well organized and easily retrievable, including test case printouts and TSFs.
- A review of previously executed test cases may be beneficial in the validation and fault isolation of a current test.
- The TSF should provide sufficient information to allow the test team to duplicate the test, if necessary.

5.5.4 Activity: Test Case Validation

Completed by: Test Team

Description: The test team analyzes the test results, validating that they fulfill the test item passage criteria and are consistent with any expected results. For any test item that does not pass, the test team initiates an SFR and returns it, along with appropriate test case printouts, to the development team for fault isolation.

Input Activities:

- Test item mapping
  - Test items organized by build
  - Passage criteria
- Test case execution
  - Test case printout
  - TSF

Output Activities:

- Fault isolation
  - Test case printout
  - SFR

Exit Criteria:

- Have all test items corresponding to the test case been evaluated?
- Have all necessary SFRs been generated?

Notes:

- The test team may use a variety of tools in the validation process, including simulators, comparison software, and hand calculations.
- The test team should pass all pertinent test case execution information to the development team to aid in the fault isolation process.

APPENDIX—BOX STRUCTURE ALGORITHM

The 11 steps used in the box structure algorithm outlined by Mills (References 6 and 10) are as follows:

1. Define Black Box Stimuli—Determine every possible stimulus for the black box.

2. Define Black Box Behavior—For every possible stimulus, determine its complete response in terms of its stimulus history.

3. Discover State Data Requirements—For each response, describe its stimulus history as a state data requirement.

4. Define the State Data—Select the subset of the required state data that is to be maintained as state data at this level.

5. Design the State Box—For the state data selected at this level, determine the internal black box required for the state box.

6. Verify the State Box—Verify the correctness of the state box with respect to the required black box behavior.

7. Discover State Data Accesses—For each item of state data and each possible stimulus, determine all possible ways of accessing the data.

8. Define Data Abstractions—Develop a schema to organize state data into data abstractions so they may be effectively accessed.

9. Design the Clear Box—Develop uses of the defined data abstractions to replace the internal black box of the state box.

10. Verify the Clear Box—Verify the correctness of the clear box with respect to the state box.

11. Repeat Stepwise Expansion Until Complete—For every new data abstraction, repeat steps 1–10 of the algorithm until acceptable program and data descriptions are reached.

GLOSSARY

BDR	Build Design Review
CDR	Critical Design Review
COF	Component Origination Form
CRF	Change Report Form
FDD	Flight Dynamics Division
GSFC	Goddard Space Flight Center
IC	integrated circuit
JCL	job control language
MTTF	mean-time-to-failure
NASA	National Aeronautics and Space Administration
NCAL	nonexecutable library
PC	personal computer
PDL	program design language
PDR	Preliminary Design Review
SEL	Software Engineering Laboratory
SFR	Software Failure Report
TBD	to be determined
TSF	Test Status Form

REFERENCES

1. SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

2. SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

3. SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

4. SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

5. SEL-86-006, "Determining Software Productivity Leverage Factors in the SEL," *Proceedings From the Eleventh Annual Software Engineering Workshop*, F. McGarry, S. Voltz, and J. Valett, December 1986

6. H. D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems," *IEEE Software*, June 1988, pp. 23–36

7. R. H. Cobb and H. D. Mills, "Engineering Software Under Statistical Quality Control," *IEEE Software*, November 1990, pp. 44–54

8. H. D. Mills, M. Dyer, and R. C. Linger, "Cleanroom Software Engineering," *IEEE Software*, September 1987, pp. 19–24

9. M. Dyer, "An Approach to Statistical Testing for Cleanroom Software Development," *IBM TR. 86.0002*, August 12, 1983

10. H. D. Mills, "Cleanroom Software Engineering," *Aerospace Software Engineering*, 1990

11. SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. McGarry, S. Waligora, et al., November 1990

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-77-004, *A Demonstration of AXES for NAVPAK*, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, *GSFC NAVPAK Design Specifications Languages Study*, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker, W. A. Taylor, et al., July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979

SEL-79-003, *Common Software Module Repository (CSMR) System Description and User's Guide*, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop*, November 1979

10000229
1113/1400

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation*, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, *Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study*, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop*, November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems*, J. F. Cook and F. E. McGarry, December 1980

SEL-80-008, *Tutorial on Models and Metrics for Software Management and Engineering*, V. R. Basili, 1980

SEL-81-008, *Cost and Reliability Estimation Models (CAREM) User's Guide*, J. F. Cook and E. Edwards, February 1981

SEL-81-009, *Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation*, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings of the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-107, *Software Engineering Laboratory (SEL) Compendium of Tools (Revision 1)*, W. J. Decker, W. A. Taylor, E. J. Smith, et al., February 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

10000229
1113/1400

SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings of the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, M. G. Rohleder, and F. E. McGarry, October 1983

SEL-82-1006, *Annotated Bibliography of Software Engineering Laboratory Literature*, L. Morusiewicz and J. Valett, November 1991

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983

SEL-83-006, *Monitoring Software Development Through Dynamic Variables*, C. W. Doerflinger, November 1983

SEL-83-007, *Proceedings of the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings of the Ninth Annual Software Engineering Workshop*, November 1984

SEL-84-101, *Manager's Handbook for Software Development (Revision 1)*, L. Landis, F. E. McGarry, S. Waligora, et al., November 1990

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

10000229
1113/1400

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr., and V. R. Basili, May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, E. Edwards, F. McGarry, and C. Antle, December 1985

SEL-85-006, *Proceedings of the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment (FDS/SDE) Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card et al., November 1986

SEL-86-006, *Proceedings of the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada® Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada® Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987

SEL-87-010, *Proceedings of the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

10000229
1113/1400

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/ Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/ Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler, K. Pumphrey, and D. Spiegel, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott and M. Stark, September 1990

SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990

SEL-90-006, *Proceedings of the Fifteenth Annual Software Engineering Workshop*, November 1990

10000229
1113/1400

SEL-91-001, *Software Engineering Laboratory (SEL) Relationships, Models, and Management Rules*, W. Decker, R. Hendrick, and J. Valett, February 1991

SEL-91-003, *Software Engineering Laboratory (SEL) Ada Performance Study Report*, E. W. Booth and M. E. Stark, July 1991

SEL-91-004, *Software Engineering Laboratory (SEL) Cleanroom Process Model*, S. Green, November 1991

SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991

SEL-91-102, *Software Engineering Laboratory (SEL) Data and Information Policy (Revision 1)*, F. McGarry, August 1991

SEL-RELATED LITERATURE

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments*. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[8]Bailey, J. W., and V. R. Basili, "Software Reclamation: Improving Post-Development Reusability," *Proceedings of the Eighth Annual National Conference on Ada Technology*, March 1990

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," *ASME Advances in Computer Technology*, January 1980, vol. 1

Basili, V. R., *Tutorial on Models and Metrics for Software Management and Engineering*. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference*, September 1985

[7]Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

[7]Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

[8]Basili, V. R., "Viewing Maintenance of Reuse-Oriented Software Development," *IEEE Software*, January 1990

10000229
1113/1400

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[9]Basili, V. R., and G. Caldiera, *A Reference Architecture for the Component Factory*, University of Maryland, Technical Report TR-2607, March 1991

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

[4]Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," *Proceedings of the ACM SIGMETRICS Symposium/ Workshop: Quality Metrics*, March 1981

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," *Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost*. New York: IEEE Computer Society Press, 1979

[5]Basili, V. R., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

[5]Basili, V. R., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

[5]Basili, V. R., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

10000229
1113/1400

[7]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

[8]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: Model-Based Reuse Characterization Schemes*, University of Maryland, Technical Report TR-2446, April 1990

[9]Basili, V. R., and H. D. Rombach, *Support for Comprehensive Reuse*, University of Maryland, Technical Report TR-2606, February 1991

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., *Comparing the Effectiveness of Software Testing Strategies*, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

[5]Basili, V. R., and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[9]Basili, V. R., and R. W. Selby, "Paradigms for Experimentation and Empirical Studies in Software Engineering," *Reliability Engineering and System Safety*, January 1991

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

[2]Basili, V. R., and D. M. Weiss, *A Methodology for Collecting Valid Software Engineering Data*, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research*, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," *Proceedings of the Second Software Life Cycle Management Workshop*, August 1978

10000229
1113/1400

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures*, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," *Proceedings of the Third International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1978

[9]Booth, E. W., and M. E. Stark, "Designing Configurable Software: COMPASS Implementation Concepts," *Proceedings of Tri-Ada 1991*, October 1991

[9]Briand, L. C., V. R. Basili, and W. M. Thomas, *A Pattern Recognition Approach for Software Engineering Data Analysis*, University of Maryland, Technical Report TR-2672, May 1991

[5]Brophy, C. E., W. W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

[5]Card, D. N., and W. W. Agresti, "Resolving the Software Science Anomaly," *The Journal of Systems and Software*, 1987

[6]Card, D. N., and W. W. Agresti, "Measuring Software Design Complexity," *The Journal of Systems and Software*, June 1988

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D. N., F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

Doubleday, D., *ASAP: An Ada Static Source Code Analyzer Program*, University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

Hamilton, M., and S. Zeldin, *A Demonstration of AXES for NAVPAK*, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

[5]Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

[5]Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[5]McGarry, F. E., and W. W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

10000229
1113/1400

National Aeronautics and Space Administration (NASA), *NASA Software Research Technology Workshop* (Proceedings), March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

[5]Ramsey, C. L., and V. R. Basili, *An Evaluation of Expert Systems for Software Engineering Management*, University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987

[8]Rombach, H. D., "Design Measurement: Some Lessons Learned," *IEEE Software*, March 1990

[9]Rombach, H. D., "Software Reuse: A Key to the Maintenance Problem," *Butterworth Journal of Information and Software Technology*, January/February 1991

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[7]Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988

[9]Seidewitz, E., "Object-Oriented Programming Through Type Extension in Ada 9X," *Ada Letters*, March/April 1991

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[9]Seidewitz, E., and M. Stark, "An Object-Oriented Approach to Parameterized Software in Ada," *Proceedings of the Eighth Washington Ada Symposium*, June 1991

[8]Stark, M., "On Designing Parametrized Systems Using Ada," *Proceedings of the Seventh Washington Ada Symposium*, June 1990

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989

[5]Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987

[8]Straub, P. A., and M. V. Zelkowitz, "PUC: A Functional Specification Language for Ada," *Proceedings of the Tenth International Conference of the Chilean Computer Science Society*, July 1990

[7]Sunazuka, T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989

Turner, C., and G. Caron, *A Comparison of RADC and NASA/SEL Software Development Data*, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, *NASA/SEL Data Compendium*, Data and Analysis Center for Software, Special Publication, April 1981

[5]Valett, J. D., and F. E. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

[5]Wu, L., V. R. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium-Scale Software Projects," *Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science*. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (Proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM*, June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

[8]Zelkowitz, M. V., "Evolution Towards Specifications Environment: Experiences With Syntax Editors," *Information and Software Technology*, April 1990

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

NOTES:

[1]This article also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I*, July 1982.

[2]This article also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983.

[3]This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

[4]This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

[5]This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

[6]This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

[7]This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.

[8]This article also appears in SEL-90-005, *Collected Software Engineering Papers: Volume VIII*, November 1990.

[9]This article also appears in SEL-91-005, *Collected Software Engineering Papers: Volume IX*, November 1991.